Preventing Insecure Direct Object References In App Development

Author: Hui Wang

Course: Computer Security

Mentor: Ming Chow

Abstract:

Insecure Direct Object References has been presented on the list of OWASP Top 10 Web application security risks since 2007. This vulnerability allows an authorized user to fetch the information of other users, and could be found in any type of software applications. It becomes an increasingly important issue in mobile security due to the prevalence of mobile apps that are gathering users' personal information, such as health and medical apps. This paper will talk about what is Insecure Direct Object References vulnerability and the examples of its exploitation, and discuss about how to detect and prevent this threat.

1 Introduction

Insecure Direct Object References is a type of prevalent vulnerability that allows requests to be made to specific objects through pages or services without the proper verification of requester's right to the content. It is mostly found in Web applications or Mobile applications. As OWASP's description, "Insecure Direct Object References occur when an application provides direct access to objects based on user-supplied input. As a result of this vulnerability attackers can

bypass authorization and access resources in the system directly, for example database records or files" [1].

By exploiting Insecure Direct Object References, attackers can bypass authorization and access resources directly by modifying the value of a parameter used to directly point to an object (i.e. by modifying the user account ID in a URL string to access the information of other users) . The potentially accessed resources can be database entries belong to other users, files in the system, and more. The references pointing to these resources, which may be exploited by attackers, can be a database key or a directory of a file. If the application takes user supplied input and uses it to retrieve an object without performing sufficient authorization checks, this vulnerability is enabled.

2 Why Care About Insecure Direct Object References?

Insecure Direct Object References has been included in OWASP Top 10 since 2007. In 2010, it was listed as the number fourth vulnerability with a prevalence of common. Although this vulnerability is easy to exploit and easy to detect, it is still usually ignored by developers when they are designing and implementing applications. Following are two typical stories showing you how prevalently this vulnerability exists in web applications, since they happened in an Internet giant's and a government's websites:

In February 2014, Insecure Direct Object References vulnerability was found in Yahoo! , the 4th most visited website on the Internet. A hacker spotted it existed in the Yahoo! sub-domain 'suggestions.yahoo.com', which could allow an attacker to delete all the posted thread and comments on Yahoo's Suggestion Board website, which are totally more than 1 million and half records [2].

June 2000, Australian Treasury GST(Goods and Services Tax) website was hacked by a computer science student. He found this vulnerability incidentally by simply hitting the wrong key when he was typing in an URL, and accessed private information from some of the other 17,000 businesses already on the GST Start-up website [3].

As we know, the issue of data security and privacy in mobile applications are becoming increasingly important due to their prevalent usage in people's daily live, and lots of mobile apps are gathering or storing users' personal data. Insecure Direct Object Reference vulnerability, which can result in information leakage, must be eliminated in mobile app development. In the new year of 2014, insecure direct object reference vulnerability was found in Snapchat allowing attackers to easily pull 4.6 million personal phone numbers out of its database. In June 2014, MyFitnessPal, a fitness app that records users' personal fitness and health data, was reported a coding error resulting in an insecure direct object

reference where anyone who was logged in was able to request and access any profile by manipulating the User ID parameter in the request [4].

3. Examples of Insecure Direct Object References Attacks

One typical example of attacks making use of this vulnerability is by modifying values of parameters in URL string. Suppose an application uses unverified data in a SQL call that is accessing account information:

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt = connection.prepareStatement(query , ...
);
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

The attacker simply modifies the ‘acct’ parameter in their browser to send whatever account number they want. If the application does not perform user verification, the attacker can access any user’s account, instead of only the intended customer’s account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Insecure direct object references vulnerability also allows attackers to manipulate records in database. In the case of Yahoo! Suggestions vulnerability, an attacker could modify the parameters in HTTP POST requests to delete 1.5 million records entered by Yahoo users. In the blog of Ibrahim Raafat, an Egyptian

Cyber Security Analyst who report this vulnerability, he descripted how he found it:

At first, he found he could delete his comments, so he opened Live HTTP Headers to check the content in the POST request:

```
prop=addressbook&fid=367443&crumb=Q4.PSLBfBe.&cid=1236547890&cmd=
delete_comment
```

Where parameter 'fid' is the topic id and 'cid' is the respective comment ID. While testing, he found changing the fid and cid parameter values allow him to delete other comments from the forum, that are actually posted by another user.
Next, he used the same method to test post deletion mechanism and found a similar vulnerability in that. A normal HTTP Header POST request of deleting a post is:

```
POST cmd=delete_item&crumb=SbWqLz.LDP0
```

He found that, appending the fid (topic id) variable to the URL allows him to delete the respective post of other users:

```
POST cmd=delete_item&crumb=SbWqLz.LDP0&fid=xxxxxxxx
```

Moreover, the attackers may find out the internal naming conventions and infer the method names for operation functionality [6]. For instance, if an application has URLs for retrieving detail information of an object like:

```
tic.com/Customers/View/2148102445
```

or

```
tic.com/Customers/ViewDetails.aspx?ID=2148102445
```

Attackers will try to use the following URLS to perform modification on the object:

```
tic.com/Customers/Update/2148102445
```

or

```
tic.com/Customers/Modify.aspx?ID=2148102445
```

or

```
tic.com/Customers/admin
```

4 The Detection of Insecure Direct Object References Vulnerability

Generally, the first step of testing this vulnerability is to “map out all locations in the application where user input is used to reference objects directly” [5]. These locations include where user input is used to access a database row, a file, application pages, etc. Secondly, modify the value of the parameter used to reference objects and assess whether it is possible to retrieve objects belonging to other users and whether authorization can be bypassed.

4.1 Static Analysis

One important approach is code review of the application and verify whether the following mechanisms are implemented safely:

(1) For direct references to restricted resources, the application needs to verify the user is authorized to access the exact resource they have requested.

(2) If the reference is an indirect reference, the mapping to the direct reference must be limited to values authorized for the current user.

4.2 Dynamic Analysis

One way to test would be by having multiple users to cover different owned objects and functions. For example, assume two users each having access to different objects and with different privileges (i.e. administrator users or normal users). Logon as one user to see whether there are direct references to objects or functionality that belong to other users. Another advantage of having multiple users is to save testing time in guessing different object names that belong to other users.

Typical scenarios for this vulnerability and the methods to test for each include:

(1) The value of a parameter is used directly to retrieve a database record.

Sample request:

```
http://foo.bar/somepage?invoice=12345
```

where the value of the “invoice” parameter is the object id used as an index to query the invoice record in the database. By modifying the value of the parameter, it is possible to retrieve any invoice record, regardless of whether the object belongs to the user.

(2) The value of a parameter is used directly to perform an operation in the system.

Sample request:

```
http://foo.bar/changepassword?user=someuser
```

where the value of the “user” parameter is used to tell the application for which user it should change the password. Typically it is the first step in the password changing wizard and leads to a page that requests new password for the specified user. This URL can be used to test whether a logged in user can open the password modification page for another user.

(3) The value of a parameter is used directly to retrieve a file system resource.

Sample request:

```
http://foo.bar/file.jsp?file=report.txt
```

where the value of the “file” parameter is used to specify what file the user intent to retrieve. By modifying this value, attackers will be able to retrieve objects belonging to other user.

Another sample request can be:

```
http://foo.bar/file.jsp?file=**../../../etc/shadow**
```

where that attacker is using directory traversal attack and attempt to retrieve the shadow file for cracking the passwords in the system.

(4) The value of a parameter is used directly to access application functionality.

Sample request:

```
http://foo.bar/accesPage?menuitem=12
```

where the value of the “menuitem” parameter is used to tell the application which menu item(and therefore which application functionality) the user wants to access. By modifying this value, a user can bypass authorization and access application functionality that are not included in the privileges of this user account.

5. The prevention of Insecure Direct Object References

From the cases and examples presented above, we can see that insecure direct object typically caused by several design flaws, such as lack of access control, using direct reference to internal object that is exposed and predictable (i.e. customer ID are easily guessed because it is integer and auto-incrementing). Generally, there are several major approaches to prevent and defense insecure direct object references attacks as following:

(1) Access Control Check:

One essential defense is to check access control. On each use of a direct object reference from an untrusted source, the application should perform an access control check to ensure the user is authorized for the requested object or service. One way to implement this is to use role-based authorization. The idea behind is associating a list of roles with a user, and the service code queries the list to make decisions about whether the user has privilege to access the requested object or service at run time.

(2) Indirect Reference Map:

An indirect reference map is a substitution of the internal reference with an alternate ID. It is used for mapping from a set of internal direct object references (i.e. database keys, filenames, etc.)to a set of indirect reference that can be safely exposed externally. In the cases of Australian Treasury GST as mentioned above, attackers can make use of the exposure of easily predictable direct references to retrieve records that do not belong to the current logged in user. The direct references are user IDs that are integer and auto-incrementing. Take this as an example, an indirect reference map can be implemented as follows:

a) Create a map on the server between that actual key, user ID in the database (i.e. 1011, 1012, …,), and the substitution, which can be a long hash value or a GUID that is easily generated but difficult to predicted by users.

b) The user ID is translated to its substitution key before being exposed to the UI.

3) After the substitution key is returned to the server, it is translated back to the original user ID before the record is retrieved.

3) Validate User Inputs:

Always check user input before using it because evil input is the root of cause of this threat. There must be validation performed in server side, since client-side validation cannot guarantee evil input to be avoided. For instance, attackers may

make use of proxy tools to capturing and reissuing HTTP requests to bypass the client-side validation.

(4) Use per user or session indirect object references:

This approach can be used to prevent attackers from directly targeting unauthorized resources. For example, use a drop down list of resources authorized instead of database key for the current user to limit the user input. The application has to map the per-user indirect reference back to the actual database key on the server [1].

6. Summary

Insecure direct object references vulnerability provides a hole for attackers to access to resources beyond their privileges. Its prevalence in web application and mobile applications results from its easy exploitability and also because it is often overlooked by developers. Actually, compared to its potential serious consequences (can compromise all of the data that can be referenced by the parameter), its easy detection and low cost in implementation tells us that prevention of this risk is must-do and easily achieved. The essential approach for the defense of insecure direct object references is access control, which is also needed by the defenses of many other types of threats. Indirect reference map, on the other hand, could be another valuable layer of defense.

References:

[1] OWASP, *"Top 10 2010-A4-Insecure Direct Object References",* 2010. [Online]. Available: https://www.owasp.org/index.php/Top_10_2010-A4-Insecure_Direct_Object_References

[2] Ibrahim Raafat, *"Vulnerability in Yahoo allowed me to delete more than 1 million and half records from Yahoo database*", February 26, 2014. [Online]. Available: http://pwnrules.com/yahoo-suggestions-vulnerability/

[3] ABC Australia, "*Computer student hacks into GST web site*", June 29, 2000. [Online]. Available: http://www.abc.net.au/7.30/stories/s146760.htm

[4] Michael Mimoso, "*Fitness App Patches Privacy Vulnerability*", September 22, 2014. [Online]. Available: http://threatpost.com/fitness-app-patches-privacy-vulnerability/108431

[5] OWASP, "*Testing for Insecure Direct Object References (OTG-AUTHZ-004)".* [Online]. Available: https://www.owasp.org/index.php/Testing_for_Insecure_Direct_Object_References_%28OTG-AUTHZ-004%29

[6] OWASP, A4 Direct Object Lecture, [Online]. Available: https://ncsu-secure-software.appspot.com/softwaresecurity/unit?unit=13&lesson=14

[7] TroyHunt*, "OWASP Top 10 for .NET developers part 4: Insecure direct object reference".* [Online]. Available: http://www.troyhunt.com/2010/09/owasp-top-10-for-net-developers-part-4.html